Social Network WebApplication

Budeme použivat - ASAP.NET + React.JS + MobX

Softwere bude cross-patform taže možeme použit akykolvek budeme chciet (od windowsu cez MacOs až po Linux)

Aplikacie -

Postman - <https://www.postman.com/>

.Net 8.0 - <https://dotnet.microsoft.com/en-us/download>

Note.js - <https://nodejs.org/en/download>

Visual Studio Code - <https://code.visualstudio.com/>

Extensions pre Visual Studio Code

* **SQLite**  ***v0.14.1***
* **NuGet Gallery**  ***v0.0.24***
* **Material Icon Theme**  ***v5.1.0***
* **C#**  ***v2.28.11***

Walking skeleton pre API -

Je to maly system ktory poskytuje end-end funkcionalitu, data budu v našej DB a budme ich zobrazovat cez našu API klientovi

[Social Network WebApplication 1](#_Toc1380891572)

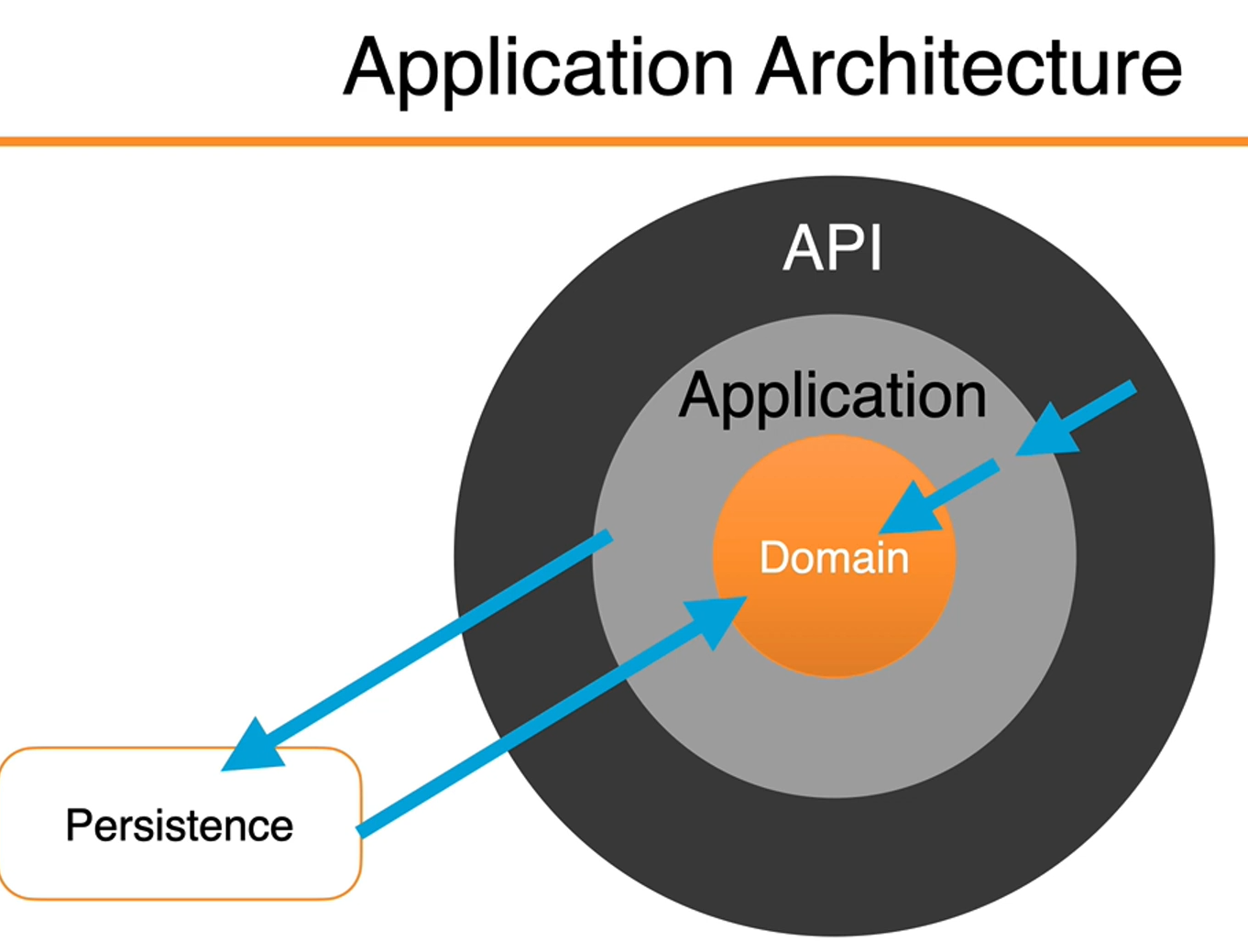
[Vytvorenie dotnet projektu 3](#_Toc1347927585)

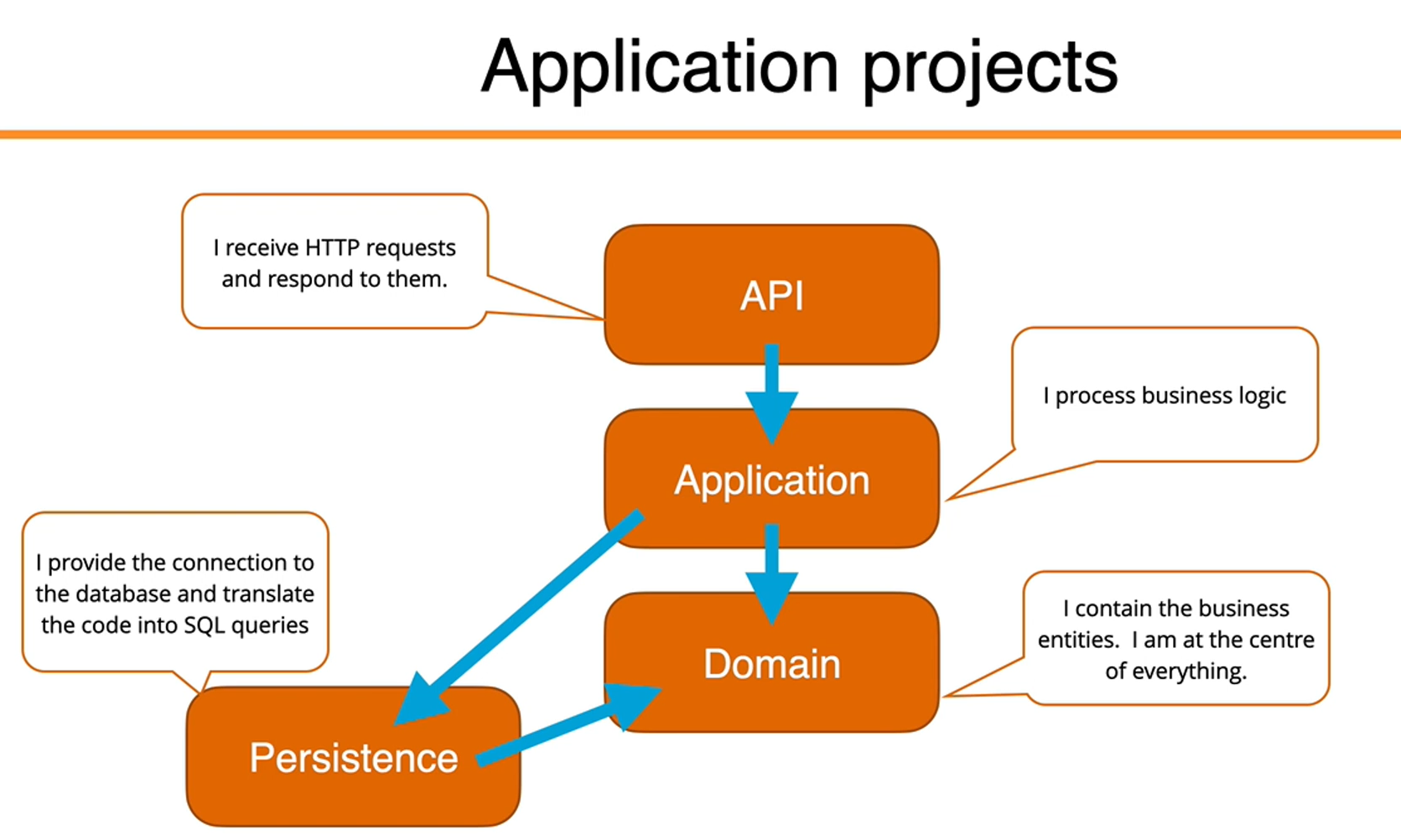
[Walking Skeleton - API 3](#_Toc1083651893)

[Domain Entity - 4](#_Toc1766627536)

[Pridanie entity do Db framework 5](#_Toc1615606911)

[Pridanie API Controllera 9](#_Toc446326001)





Vytvorenie dotnet projektu

# Walking Skeleton - API

V CMD zadame -

**dotnet new list**

Tento príkaz je užitočný, ak si nie ste istí, ktorú šablónu použiť pre svoj projekt, alebo ak chcete preskúmať dostupné možnosti pred vytvorením nového projektu. Poskytne vám rýchly prehľad šablón spolu s ich krátkymi názvami, podporovanými jazykmi a značkami, aby vám pomohol vybrať vhodnú šablónu pre vaše potreby.

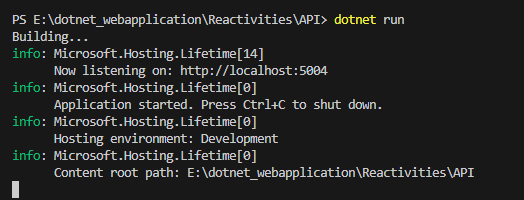
Scriptom - “create-projects-win.ps1”

Tento skript je skript prostredia PowerShell, ktorý sa používa na nastavenie základnej štruktúry projektu .NET pre aplikáciu web API spolu s podpornými knižnicami tried pre aplikačnú logiku, modely domény a perzistenciu údajov.

**dotnet run -**

(Tento príkaz sa bežne používa počas vývoja na rýchlu kompiláciu a testovanie aplikácií .NET bez toho, aby ste museli ručne spúšťať viacero príkazov na zostavenie a spustenie. Je obzvlášť užitočný pre konzolové aplikácie a webové aplikácie ASP.NET Core počas fázy vývoja.)

Otestujeme či naš projekt sa zostavil spravne



**Postman**

Nam bude služit na testovanie našje apliakcie (može sa použit aj swagger)

Pre dalšim pisanim kodu netreba zabudnut v configu nastavit

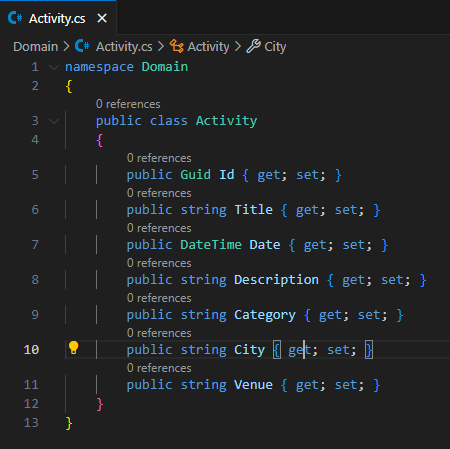
“**<Nullable>disable</Nullable>**”

je prvok XML, ktorý sa používa v súbore projektu alebo v konfiguračnom súbore na určenie toho, že nulovateľné referenčné typy sú pre daný projekt alebo jeho časť zakázané.

Domain Entity -

doménová entita sa zvyčajne vzťahuje na triedu alebo štruktúru, ktorá predstavuje koncept alebo entitu v rámci domény vašej aplikácie. "Doména" sa v tomto kontexte vzťahuje na problémový priestor alebo predmet, ktorým sa vaša aplikácia zaoberá.

Doménové entity sa často používajú pri návrhu riadenom doménou (Domain-Driven Design - DDD), čo je prístup k vývoju softvéru, ktorý kladie dôraz na modelovanie problémovej domény v softvéri, pričom sa zameriava na logiku a správanie domény.



Classa acitivity služi na getovanie a setovanie value ktore sa v nej nachadzaju (pre dalšie použivanie).

## Pridanie entity do DB framework

public class DataContext : DbContext

Táto syntax naznačuje, že trieda DataContext dedí od triedy DbContext (alebo ju rozširuje). V objektovo orientovanom programovaní dedičnosť umožňuje novej triede (odvodenej alebo dcérskej triede) dediť vlastnosti a správanie z existujúcej triedy (základnej alebo rodičovskej triedy).

Danej tirede vytovirme Constructor

public DataContext(DbContextOptions options) : base(options)

{

}

V kontexte konštruktora public DataContext(DbContextOptions options) : base(options) je DbContextOptions trieda v Entity Framework Core, ktorá sa používa na konfiguráciu správania DbContextu. Tento konštruktor sa zvyčajne používa, keď chcete vytvoriť inštanciu vlastnej triedy DbContext.

Options - Tento parameter je inštanciou DbContextOptions. Predstavuje konfiguračné nastavenia, ktoré bude DataContext používať.

Do Porgram.cs

Sme pridali conenction string lebo DB potrebuje conneciton string

builder.Services.AddDbContext<DataContext>(opt =>

{

opt.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection"));

}

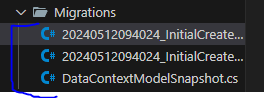
);

Dalej staihneem dotnet of - <https://www.nuget.org/packages/dotnet-ef>

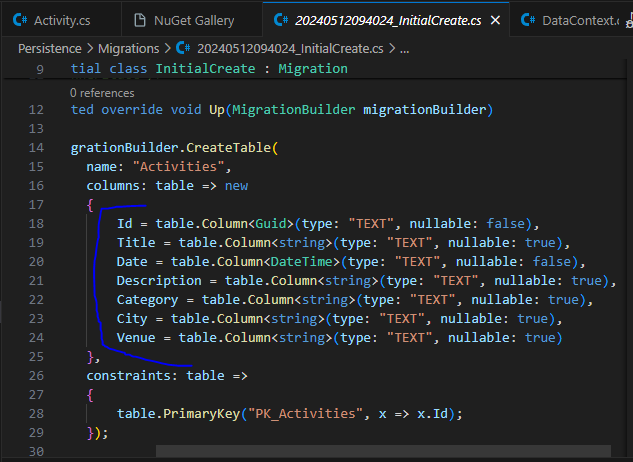
Ktory nam pomože s migraciou našho codu do DB

PS E:\dotnet\_webapplication\Reactivities> dotnet ef migrations add InitialCreate -s API -p Persistence

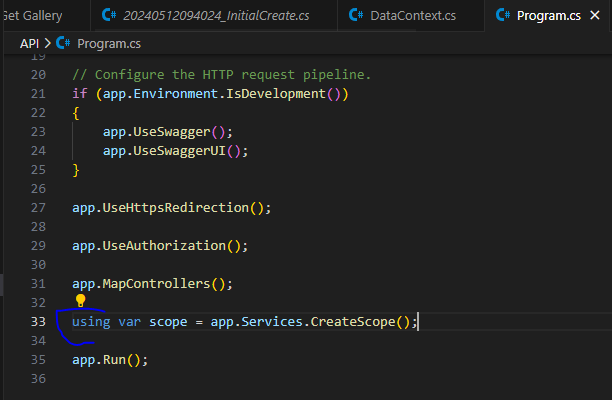
Ak sa pozrieme vytovirlo nam to 3 scripty



Hned v prvom si vieme skontrolovat stplce ktore nam to vytvori na zaklade prop activities ktore sme vytvorili (getere a setere)

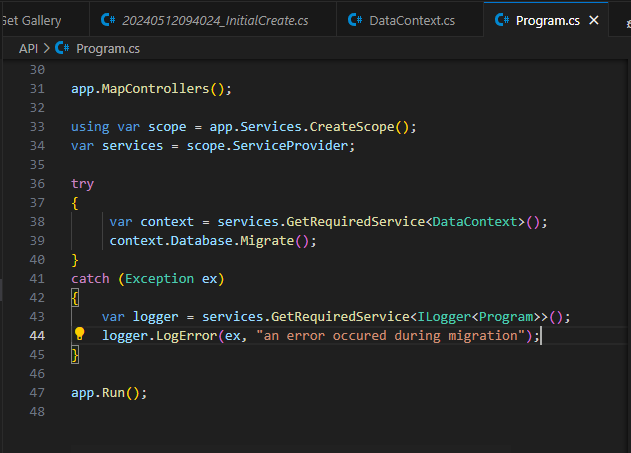
Create database

V clasee Program.cs red spustenym vytvorime



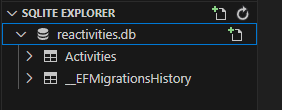
Takže keď vykonáte príkaz var scope = app.Services.CreateScope();, vytvoríte nový scope v kontajneri služieb aplikácie, ktorý potom môžete použiť na riešenie služieb špecifických pre tento scope. Toto sa často používa, keď potrebujete vyriešiť služby v rámci konkrétneho kontextu alebo rozsahu, napríklad v rámci požiadavky v aplikácii ASP.NET Core.

Pridame try catch pre migraciu (ak by počas toho nastal problem)



Potom je fajn použit dotnet watch

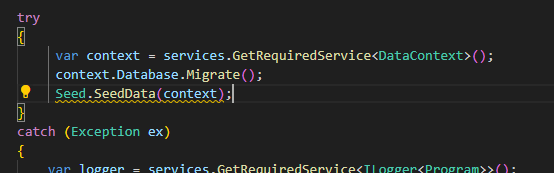
je nástroj príkazového riadka poskytovaný v súbore .NET Core SDK, ktorý umožňuje automatické prebudovanie a reštartovanie aplikácie .NET Core po zistení zmien v zdrojových súboroch. Je obzvlášť užitočný počas vývoja na urýchlenie procesu vývoja tým, že eliminuje potrebu ručného zastavenia, prebudovania a opätovného spustenia aplikácie po vykonaní zmien.



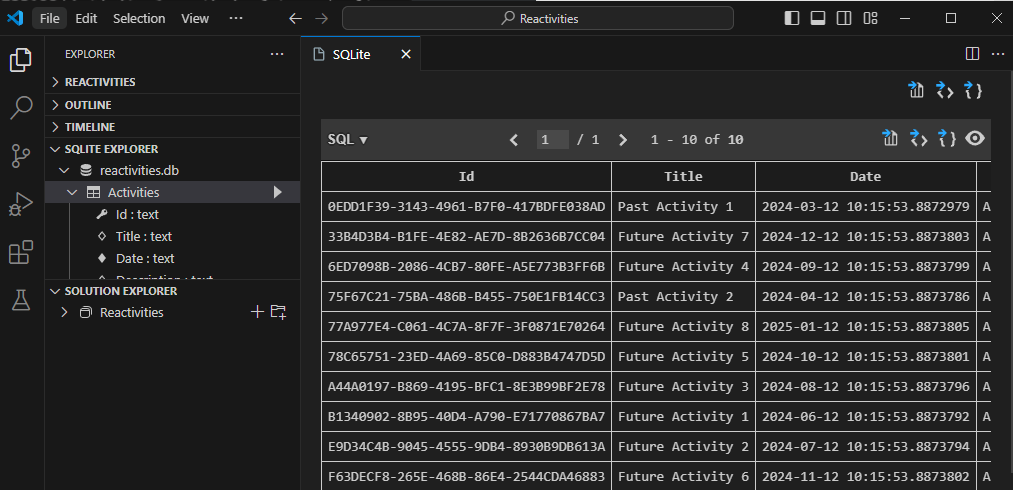
Toto použivame len pre development aby sme videli že to funguje

Seedovanie data je vytovrenie testovacih dat , na tot možeme použi “SeedData.txt”v ktorom sa nachazda cela trieda s tymito datami

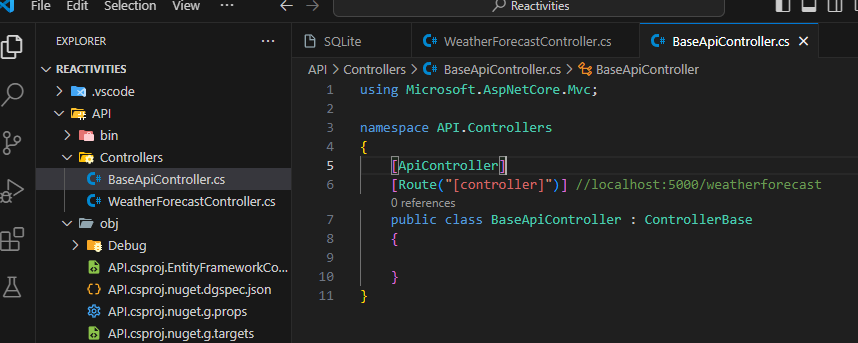
Po migrovani DB rovno do nej hodiem aj tieto Seedovane DATA (toto sme menili v program.cs)



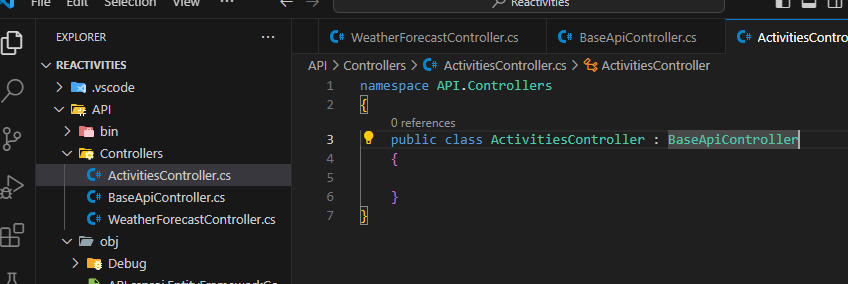
Skontrolovat si to možeme tak že vidime potom že v našej DB sa tieto udaje nachadzaju



## Pridanie API Controllera

Tym že už mame vytvoreny jeden API contorller tak sme z neho zobrali priamrne časti s tym že dedili sme z ControllerBase

Keď teda vytvoríte novú triedu kontroléra API, ktorá dedí z triedy BaseApiController, automaticky zdedí atribút [ApiController] a šablónu trasy určenú pomocou [Route("[controller]")]. To pomáha udržiavať kód DRY (Neopakuj sa) centralizovaním spoločného správania a konfigurácie kontroléra API.

Tento dedi z našho teraz vytovrnee controllera